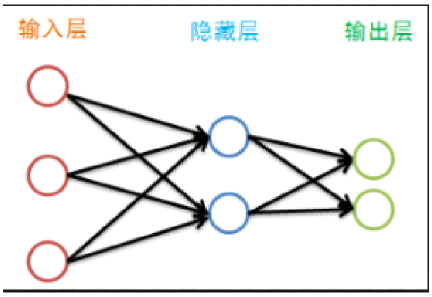
深度学习

2017年12月19日　　19:16

简介

1. 结构：中间的隐藏层是无数个机器决定的参数



2. 分类：

   standard NN：单层的神经网络

   CNN：卷积神经网络，用于图片分析

   RNN：序列神经网络，用于音频、语言等序列

   RNNs：多序列的神经网络

3. 发展的原因：

   数据量的增长 -> 机器学习 -> 数据量的进一步增长 -> 机器学习不再提升，故采用参数更多的深度学习 -> 对计算速度的要求升高 -> 算法上的改进

数据记录方式

X：图片的基础色可分为3个色：红、绿、蓝，将每个像素点上的亮度均表示出来，得到64*64的3个矩阵，表示为一个64*6*3长度的向量，X∈R^{nx}

$$x = \begin{bmatrix} \left.\begin{matrix}255\\231\\42\\\vdots\end{matrix}\right\} \text{red} \\ \left.\begin{matrix}255\\134\\202\\\vdots\end{matrix}\right\} \text{green} \\ \left.\begin{matrix}255\\134\\93\\\vdots\end{matrix}\right\} \text{blue} \end{bmatrix}$$

Y：{0,1}

训练集：{(x(1),y(1)),(x(1),y(1))......(x(m),y(m))}, m个样本

矩阵表示：

```
[.      .     .    .  ]
[.      .     .    .  ]
[X(1)X(2)    . X(m)]      [y(1),y(2),......y(m)]
[.      .     .    .  ]
[.      .     .    .  ]
  X.shape = (Nx,m)          Y.shape = (1,m)
```

逻辑回归

1. 目的：解决二分类问题
2. 函数：

   $s = \sigma(w^T x + b) = \sigma(z) = \frac{1}{1+e^{-z}}$

3. Loss function：
   - 正常的损失函数如下，但是该函数是一个非凸的函数，无法利用梯度下降法求解，也没有最优解

     $L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$

   - 故修改逻辑回归的损失函数如下：

     $L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)}\log(\hat{y}^{(i)}) + (1 - y^{(i)})\log(1 - \hat{y}^{(i)})$

$$L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})$$

设置成该形式的原因如下:

- 当y = 1 时，p(y|x) = $\hat{y}$；当y = 0 时，p(y|x) = 1-$\hat{y}$
- 将以上2个式子合为1个式子，有：p(y|x) = $\hat{y}^y(1-\hat{y})^{1-y}$
- 两边分别取log（原因：log是单调函数，保证p(y|x)单调）：log(p(y|x)) = ylog($\hat{y}$) + (1-y)log(1-$\hat{y}$)
- 因概率是最大化，损失函数是最小化，故前面添加负号：L = - (ylog($\hat{y}$) + (1-y)log(1-$\hat{y}$))

效果：

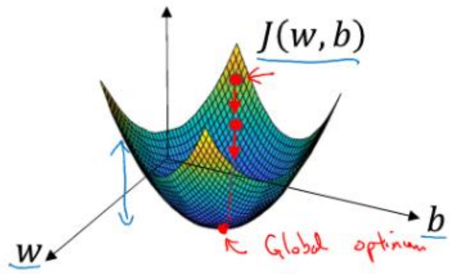
- 当y = 1时，L =- log($\hat{y}$) 应该要比较小， log($\hat{y}$)应该要比较大，$\hat{y}$应该要比较大，$\hat{y}$最大就是1
- 当y = 0时，L =- log(1-$\hat{y}$) 应该要比较小， log(1-$\hat{y}$)应该要比较大，1-$\hat{y}$应该要比较大，$\hat{y}$应该要比较小，$\hat{y}$最小就是0

4. Cost function：（平均的损失函数）

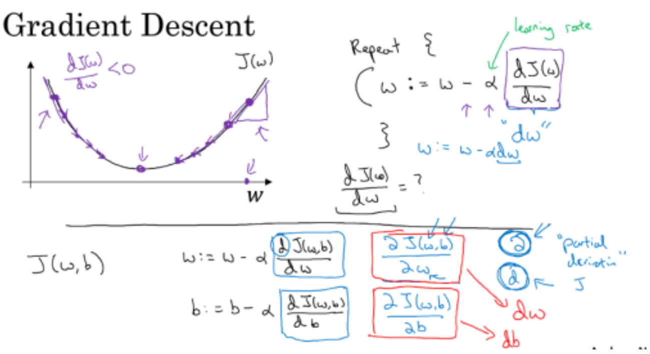
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}[(y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})]$$

梯度下降法

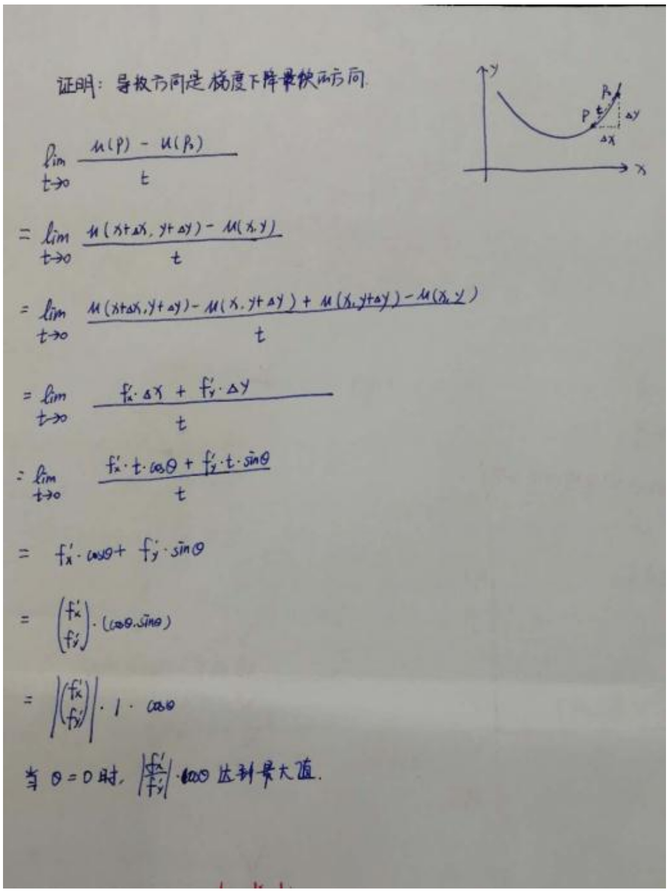
1. 希望找到w、b使得J最小



2. w、b给定初始值：此处因为是凸函数，初始值给什么样子的最后基本都会收敛到最优值的
3. 迭代：
   - W := w - a $\frac{a(J(w,b))}{aw}$ b := b - a $\frac{a(J(w,b))}{ab}$
   - 当w是一个比较大的初始值时，梯度为正，a是正数，新的w会比旧的w小，逐步减小直到最优值;
   - 当w是一个比较小的初始值时，梯度为负，a是正数，新的w会比旧的w大，逐步增大直到最优值;



4. 为什么导数的方向是梯度下降最快的方向



证明：导数方向是梯度下降最快的方向.

$$\lim_{t\to 0}\frac{u(P)-u(P_0)}{t}$$

$$=\lim_{t\to 0}\frac{u(x+\Delta x, y+\Delta y)-u(x,y)}{t}$$

$$=\lim_{t\to 0}\frac{u(x+\Delta x,y+\Delta y)-u(x,y+\Delta y)+u(x,y+\Delta y)-u(x,y)}{t}$$

$$=\lim_{t\to 0}\frac{f'_x\cdot\Delta x+f'_y\cdot\Delta y}{t}$$

$$=\lim_{t\to 0}\frac{f'_x\cdot t\cdot\cos\theta+f'_y\cdot t\cdot\sin\theta}{t}$$

$$=f'_x\cdot\cos\theta+f'_y\cdot\sin\theta$$

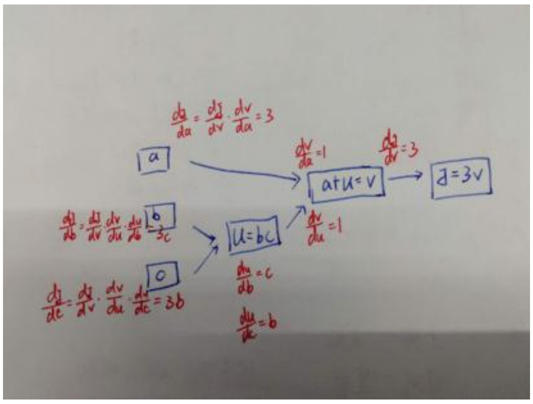
$$=\begin{pmatrix}f'_x\\ f'_y\end{pmatrix}\cdot(\cos\theta,\sin\theta)$$

$$=\left|\begin{pmatrix}f'_x\\ f'_y\end{pmatrix}\right|\cdot 1\cdot\cos\theta$$

当 $\theta=0$ 时，$\left|\begin{pmatrix}f'_x\\ f'_y\end{pmatrix}\right|\cdot\cos\theta$ 达到最大值.

正向、反向传播介绍

1. 代数例子
    - 从左到右：计算过程
    - 从右到左：求导过程。根据最终的导数可判断a、b、c变化时对J的影响，从最小化 J



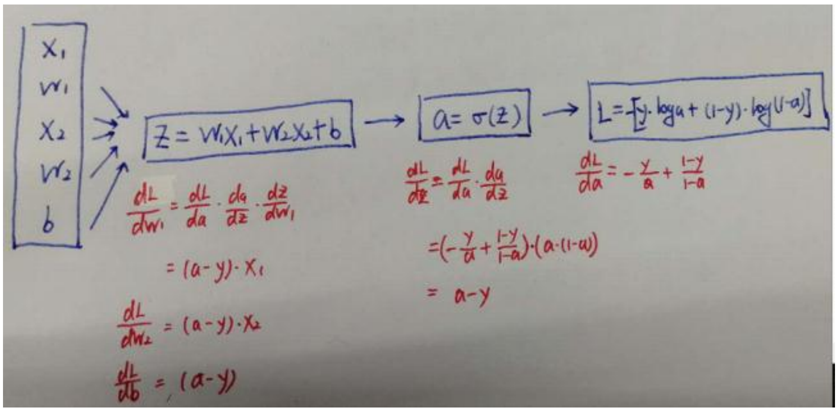
2. 逻辑回归例子
    - 理论回顾

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z)$$

$$\mathcal{L}(a,y) = -(y\log(a) + (1-y)\log(1-a))$$

- 从左到右：计算过程
- 从右到左：求导过程



- 梯度下降参数做自更新

  w1 := w1-a$\frac{\partial L}{\partial w1}$　W2 := w2-a$\frac{\partial L}{\partial w2}$　b := b-a$\frac{\partial L}{\partial b}$

- 扩展到多样本的情况

$$J(w,b)=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})=-\frac{1}{m}\sum_{i=1}^{m}[(y^{(i)}\log(\hat{y}^{(i)})+(1-y^{(i)})\log(1-\hat{y}^{(i)})]$$

  则分别求出每一个样本的$\frac{\partial L}{\partial w1}$，$\frac{\partial L}{\partial w2}$，最后m个样本做平均得到$\frac{\partial J}{\partial w1}$，$\frac{\partial J}{\partial w2}$

  梯度下降参数做更新：w1 := w1-a$\frac{\partial J}{\partial w1}$　W2 := w2-a$\frac{\partial J}{\partial w2}$　b := b-a$\frac{\partial J}{\partial b}$

向量化

1. 向量化是一种思想，要时时记得不要用for循环处理数据，多用矩阵
2. 逻辑回归梯度下降的算法实现

```
# 参数初始设定
J = 0
dw1 = 0
dw2 = 0
db = 0

# m个样本，n个特征求和
for i in range(1,m):
    z[i] = w1*x1[i] + w2*x2[i] + b
    a[i] = sigmod(z[i])
    J += -(y[i]*log(a[i]) + (1-y[i])*log(1-a[i]))  # J是m个样本的和
    dz[i] = a[i] - y[i]
    for j in range(1,n):
        dw_j += x[i]_j * dz[i]_j  # 计算J对x1的偏导数，是m个样本的

# 求平均值
J = J/m
dw1 = dw_1/m
dw2 = dw_2/m
db = db/m

# 参数更新
w1 := w1 - adw1
w2 := w2 - adw2
b := b - adb
```

3. 算法优化（利用向量化）

```
# 参数初始设定
J = 0
dW = np.zeros((n,1))
db = 0

# m个样本，n个特征求和
Z = WT * X + b
A = sigmod(Z)
dZ = A - Y
dW = X * dZ / m
db = np.sum(dZ) / m
# 参数更新
W := W - adW
b := b - adb
```

$$\boxed{db} = \frac{1}{m}\sum_{i=1}^{m} dz^{(i)}$$
$$= \frac{1}{m}\ \underline{np.\,sum}(dZ)$$
$$\boxed{dw = \frac{1}{m} X\, dz^{T}}$$
$$= \frac{1}{m}\left[\begin{matrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{matrix}\right]\left[\begin{matrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{matrix}\right]$$
$$= \frac{1}{m}\left[\underline{x^{(1)}dz^{(1)}} + \cdots + \underline{x^{(m)}dz^{(m)}}\right]$$
$$n \times 1$$

4. 更多向量化的例子

```
u = [1,2,3,4]
v = [6,6,4,1]
np.dot(u,v)
np.exp(v)      # e的v次方
np.log(v)
np.abs(v)
np.maxmun(v,0)
v ** 2
```

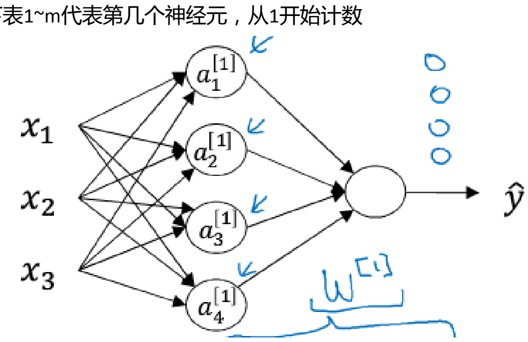
5. 广播
   - 一个序列（一个矩阵） +-*/ 一个元素
   - 一个矩阵 +-*/ 一个序列
   - `cal = A.sum(axis = 0)` # 求行和

     `percentage = A *100 /cal.reshape(1,4)` # 每个行除以行和
6. Python 技巧
   - Python中尽量使用矩阵形式的数据（5,1），不要使用数组形式（5，）

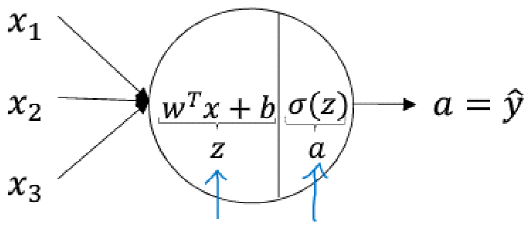
```
In [262]: import numpy as np

          a = np.random.randn(5)
          a.shape

Out[262]: (5,)

In [263]: a = np.random.randn(5,1)
          a.shape

Out[263]: (5, 1)
```

浅层神经网络

1. 二层神经网络（输入层为第0层，隐藏层为第1层，输出层为第2层）
   - 上标[ ]代表第几层，从0开始计数
   - 下表1~m代表第几个神经元，从1开始计数



2. 单个样本的二层神经网络计算过程
   - 第一层计算过程

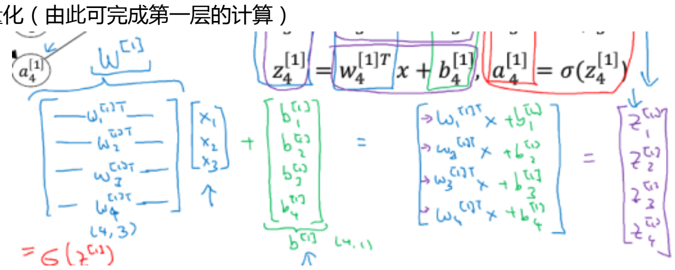


$$z = w^T x + b$$

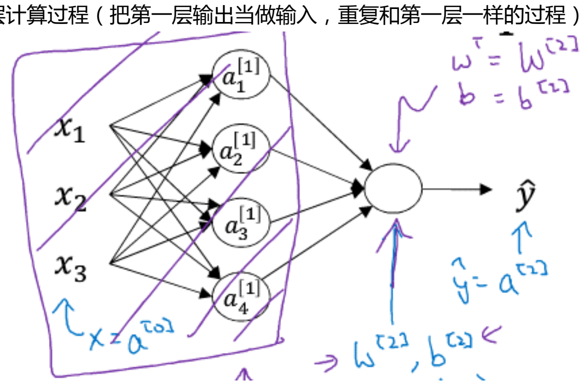
$$a = \sigma(z)$$

$$\begin{array}{l}
z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]}) \\
z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]}) \\
z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]}) \\
z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})
\end{array}$$

- 向量化（由此可完成第一层的计算）



- 第二层计算过程（把第一层输出当做输入，重复和第一层一样的过程）



- 总体向量化

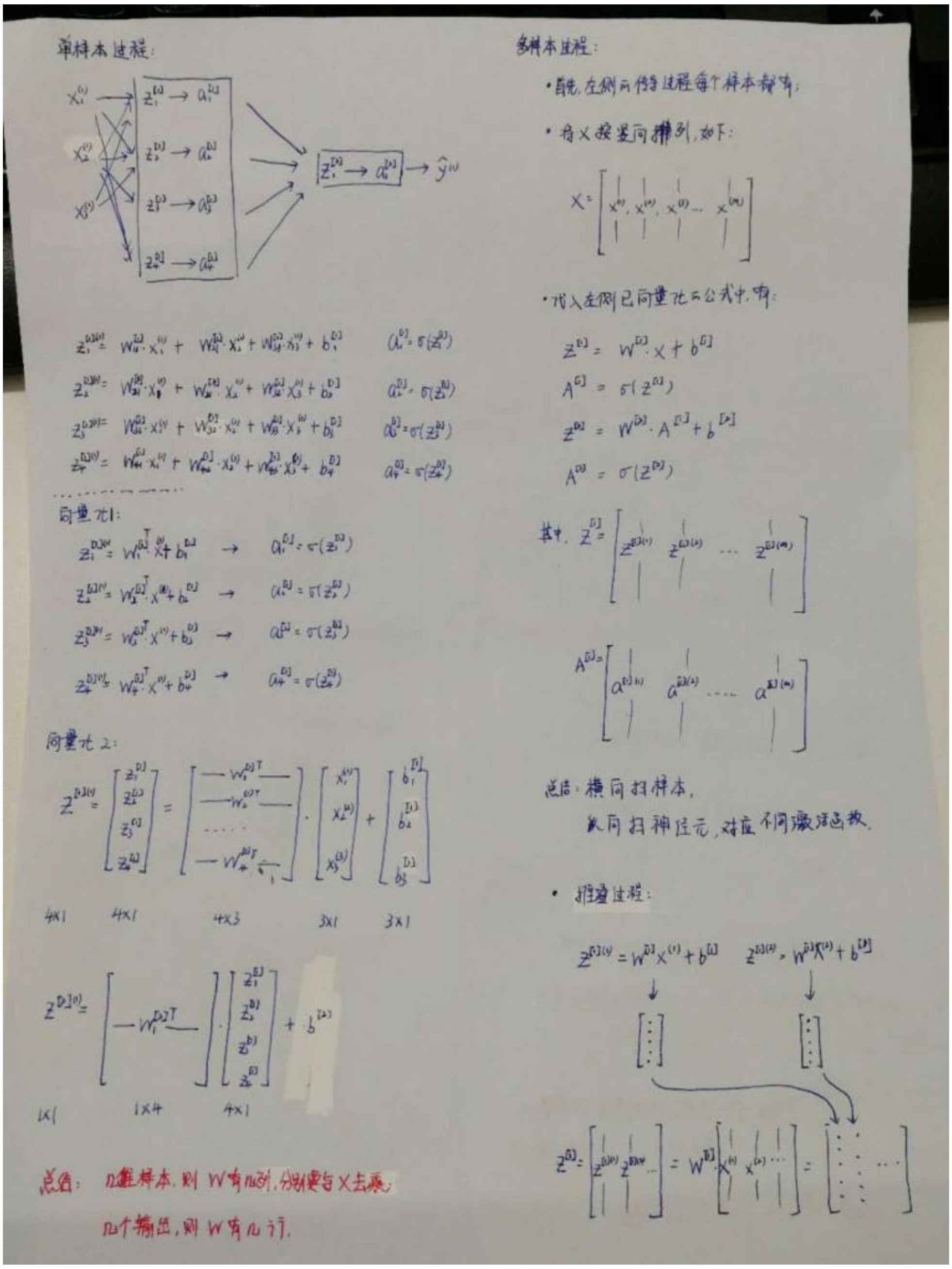
$$\underset{(4,1)}{z^{[1]}} = \underset{(4,3)}{W^{[1]}} \underset{(3,1)}{\overset{a^{[0]}}{\cancel{x}}} + \underset{(4,1)}{b^{[1]}}$$

$$\underset{(4,1)}{a^{[1]}} = \sigma(\underset{(4,1)}{z^{[1]}})$$

$$\underset{(1,1)}{z^{[2]}} = \underset{(1,4)}{W^{[2]}} \underset{(4,1)}{a^{[1]}} + \underset{(1,1)}{b^{[2]}}$$

$$\underset{(1,1)}{a^{[2]}} = \sigma(\underset{(1,1)}{z^{[2]}})$$

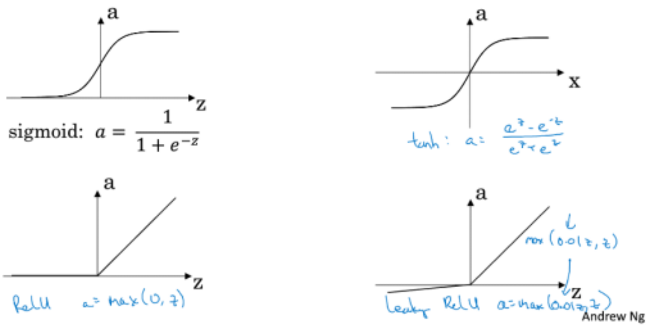
3. 多样本的二层神经网络计算过程



4. 激活函数的选择
    - 大背景：现实生活中，sigmoid函数表现总是不太好，基本不会使用；
        - 唯一例外：当二分类的最后一层输出层时可以使用

    - 为何要用非线性的激活函数呢？
        - 若激活函数是线性的，那最终的输出层还只是输入层的线性组合，不如直接去掉隐藏层，隐藏层此时不起任何作用。
          唯一例外：回归问题，在输出层要用恒等激活函数（线性激活函数），才能输出相应的数值

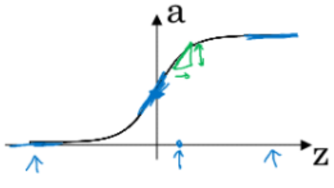
    - 更多的激活函数可以选择



        - tanh函数的优点：将sigmiod函数向下平移，使有0均值，相当于正则化；

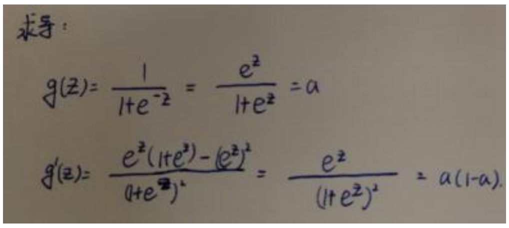
- relu函数的优点：当x趋于无穷小（或无穷大）时，导数很小，收敛速度很慢，relu不会有这种担忧
- Leaky relu的优点：当x小于0时，为微小的负值

- 激活函数的梯度下降
  - sigmoid函数



$$g(z) = \frac{1}{1+e^{-z}}$$

$a = g(z) = \frac{1}{1+e^{-z}}$

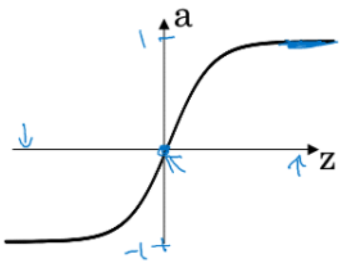


求导：

$$g(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z} = a$$

$$g'(z) = \frac{e^z(1+e^z)-(e^z)^2}{(1+e^z)^2} = \frac{e^z}{(1+e^z)^2} = a(1-a).$$

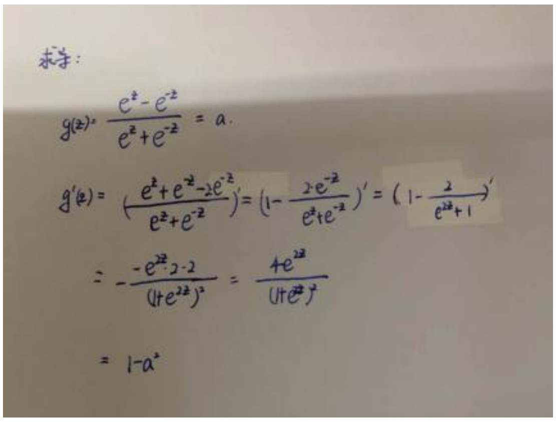
  - tanh函数



$$g(z) = \tanh(z)$$

$$= \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$|z| = 10$



求导：

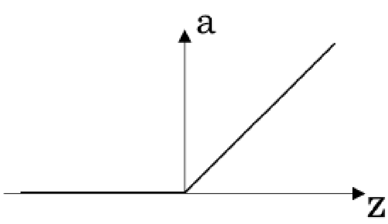
$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = a.$$

$$g'(z) = \left(\frac{e^z + e^{-z} - 2e^{-z}}{e^z + e^{-z}}\right)' = \left(1 - \frac{2e^{-z}}{e^z + e^{-z}}\right)' = \left(1 - \frac{2}{e^{2z}+1}\right)'$$

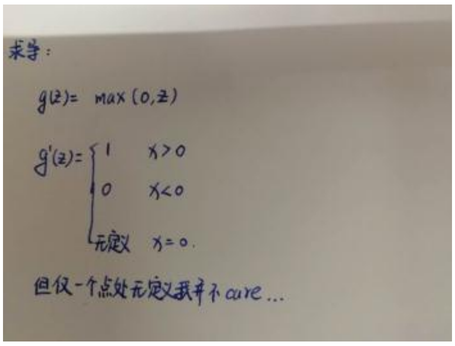
$$= -\frac{-e^{2z} \cdot 2 \cdot 2}{(1+e^{2z})^2} = \frac{4e^{2z}}{(1+e^{2z})^2}$$

$$= 1-a^2$$

  - relu函数



ReLU



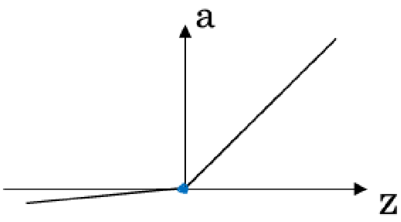
求导:

$g(z) = \max(0, z)$

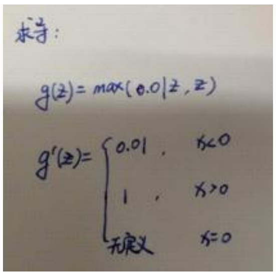
$$g'(z) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \\ \text{无定义} & x = 0. \end{cases}$$

但仅一个点处无定义我并不care...

- Leaky relu



Leaky ReLU



求导:

$g(z) = \max(0.01z, z)$

$$g'(z) = \begin{cases} 0.01, & x < 0 \\ 1, & x > 0 \\ \text{无定义} & x = 0 \end{cases}$$

- 神经网络的梯度下降
  - 正向过程

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]}) \leftarrow$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$$

  - 反向过程

$$dz^{[2]} = a^{[2]} - y$$
$$dW^{[2]} = dz^{[2]}a^{[1]^T}$$
$$db^{[2]} = dz^{[2]}$$
$$\underbrace{dz^{[1]}}_{(n^{[1]},1)} = W^{[2]T}dz^{[2]} * g^{[1]}{}'(z^{[1]})$$
$$dW^{[1]} = dz^{[1]}x^T$$
$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m}dZ^{[2]}A^{[1]^T}$$
$$J(\ldots) = \frac{1}{m}\sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$$
$$db^{[2]} = \frac{1}{m}np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$
elementwise product
$$\underbrace{dZ^{[1]}}_{(n^{[1]},m)} = \underbrace{W^{[2]T}dZ^{[2]}}_{(n^{[1]},m)} * \underbrace{g^{[1]}{}'(Z^{[1]})}_{(n^{[1]},m)}$$
$$dW^{[1]} = \frac{1}{m}dZ^{[1]}X^T$$
$$db^{[1]} = \frac{1}{m}np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

  - 证明过程

正向过程：

$$z^{[1]} = W^{[1]} \cdot X + b^{[1]}$$

$$A^{[1]} = g(z^{[1]})$$

$$z^{[2]} = W^{[2]} \cdot A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(z^{[2]})$$

输出层会用sigmoid函数.

$$L = y \cdot \log \hat{y} + (1-y) \cdot \log(1-\hat{y})$$

$$= y \cdot \log a + (1-y) \cdot \log(1-a)$$

反向过程：

$$\because \frac{\partial L}{\partial a} = \frac{y}{a} - \frac{1-y}{1-a}$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} = \left(\frac{y}{a} - \frac{1-y}{1-a}\right) \cdot \left(\frac{1}{1+e^{-x}}\right)' = a - y$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w} = (a-y) \cdot x$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial b} = a - y$$

$$\therefore dz^{[2]} = A^{[2]} - Y \qquad (n^{[2]}, 1)$$

$$dW^{[2]} = \frac{1}{m}(A-Y) \cdot A^{[1]T} = \frac{1}{m} dZ^{[2]} \cdot A^{[1]T} \qquad (n^{[2]}, n^{[1]})$$

$$db^{[2]} = \frac{1}{m} np.sum(A-Y,\ axis=1,\ keepdim=True) \qquad (n^{[2]}, 1)$$

$$dA^{[1]} = W^{[2]T} \cdot dz^{[2]} = W^{[2]T}(A-Y) \qquad (n^{[1]}, 1)$$

$$dz^{[1]} = dA^{[1]} \cdot g'(z^{[1]}) \qquad (n^{[1]}, 1)$$

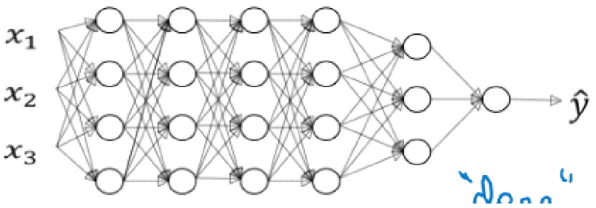
$$dW^{[1]} = \frac{1}{m} dz^{[1]} \cdot X^T \qquad (n^{[1]}, n^{[0]})$$

$$db^{[1]} = \frac{1}{m} np.sum(dz^{[1]},\ axis=1,\ keepdims=True) \qquad (n^{[1]}, 1)$$

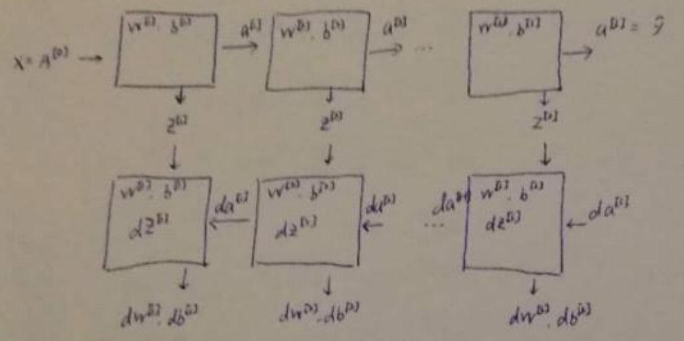
5. 神经网络参数随机化
   - 参数一定要随机化，因为若所有参数均设为0，那么所有的层和单元都在做一样的运算，浪费了很多单元
   - 随机化的起始参数一定要非常小，因为大的话，tanh、sigmoid函数的导数接近0，梯度下降收敛很慢

• 深度神经网络

1. 深度神经网络
   - 层数用L标记



   - 为什么要用深度神经网络？
     i. 神经网络工作时，浅层的单元通常做的工作是做边缘性的识别，深层的才做组装，所以深层的才更有效；
     ii. 深层的神经网络计算量更小
2. 神经网络的工作过程



$$L = y \cdot \log \hat{y} + (1-y) \cdot \log(1-\hat{y})$$

$$\frac{dL}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$= a^{[l]}$$

$$= \left[-\frac{y^{(1)}}{a^{(1)}} + \frac{1-y^{(1)}}{1-a^{(1)}}, \cdots -\frac{y^{(m)}}{a^{(m)}} + \frac{1-y^{(m)}}{1-a^{(m)}} \right]$$

$$W^{[l]} = W^{[l]} - \alpha dW^{[l]}$$

$$b^{[l]} = b^{[l]} - \alpha db^{[l]}$$

前向传播：输入 $A^{[l-1]}$，得到 $A^{[l]}$

$$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

反向传播：输入 $dA^{[l]}$，得到 $dA^{[l-1]}, dW^{[l]}, db^{[l]}$

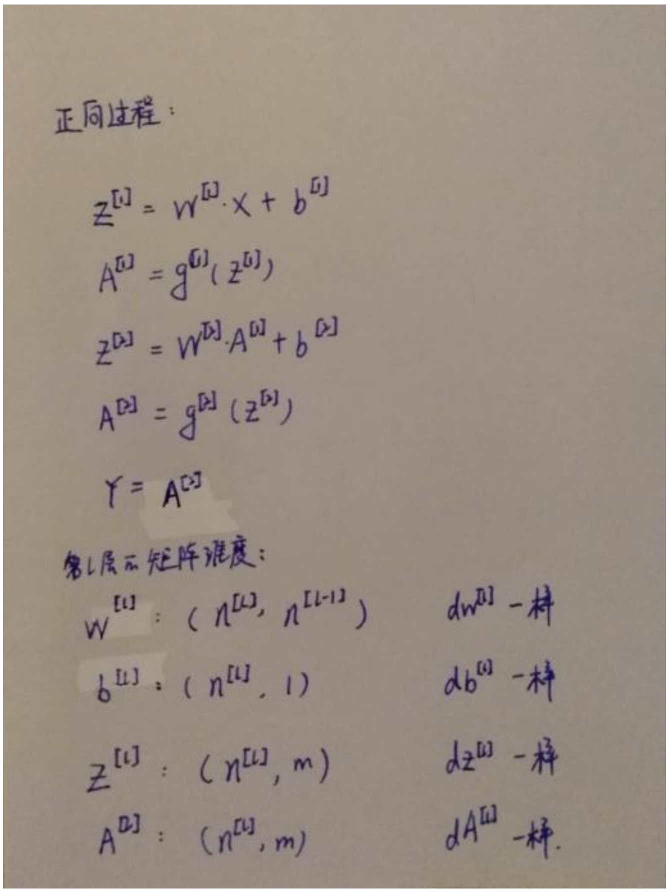
$$dz^{[l]} = da^{[l]} \cdot g^{[l]\prime}(z^{[l]}) \qquad dZ^{[l]} = dA^{[l]} \cdot g^{[l]\prime}(Z^{[l]})$$

$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]} \qquad dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]}$$

$$db^{[l]} = dz^{[l]} \qquad db^{[l]} = \frac{1}{m} \text{np.sum}(dZ^{[l]}, \text{axis}=1, \text{keepdims}=\text{True})$$

$$da^{[l-1]} = (W^{[l]})^T \cdot dz^{[l]} \qquad dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

3. 矩阵维度的介绍


正向过程：

$$Z^{[1]} = W^{[1]} \cdot X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]} \cdot A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]})$$

$$Y = A^{[2]}$$

第l层的矩阵维度：

$W^{[l]}$: $(n^{[l]}, n^{[l-1]})$ $dW^{[l]}$ 一样

$b^{[l]}$: $(n^{[l]}, 1)$ $db^{[l]}$ 一样

$Z^{[l]}$: $(n^{[l]}, m)$ $dZ^{[l]}$ 一样

$A^{[l]}$: $(n^{[l]}, m)$ $dA^{[l]}$ 一样.

4. 参数和超参数
   - 参数：W、b
   - 超参数：会影响和决定参数值的参数
     - 学习率a
     - 神经网络的层数 L
     - 每层的神经单元数 n
     - 每层使用的激活函数 g
     - 算法迭代次数